GALAHAD CHECK

USER DOCUMENTATION GALAHAD Optimization Library version 5.1

1 SUMMARY

This package uses finite difference approximations to **check the gradient of an objective function $f(\mathbf{x})$, the Jacobian matrix of a constraint function $\mathbf{c}(\mathbf{x})$, and the second derivative Hessian matrix of the Lagrangian function** $L(\mathbf{x},\mathbf{y}) = f(\mathbf{x}) - \mathbf{c}(\mathbf{x})^T\mathbf{y}$. These quantities are typically associated with a nonlinear optimization problem

$$\text{minimize } f(\mathbf{x})$$

subject to the general linear constraints

$$a_i^l \leq \mathbf{a}_i^T\mathbf{x} \leq a_i^u, \quad i = 1,\ldots,m_a,$$

general nonlinear constraints

$$c_i^l \leq c_i(\mathbf{x}) \leq c_i^u, \quad i = 1,\ldots,m,$$

and simple bound constraints

$$x_j^l \leq x_j \leq x_j^u, \quad j = 1,\ldots,n,$$

where the vectors $\mathbf{a}_i$, $\mathbf{a}^l$, $\mathbf{a}^u$, $\mathbf{c}^l$, $\mathbf{c}^u$, $\mathbf{x}^l$, and $\mathbf{x}^u$ are given, and the vectors $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^m$ are known as the primal and dual (Lagrange multiplier) vectors, respectively. The user may choose to perform a "cheap" verification of the requested derivatives, or a more detailed and "expensive" check. Function values can be supplied via internal subroutine evaluation or reverse communication.

ATTRIBUTES — Versions: `GALAHAD_CHECK_single` and `GALAHAD_CHECK_double`. **Uses:** `GALAHAD_SYMBOLS`, `GALAHAD_SPECFILE`, `GALAHAD_SPACE`, `GALAHAD_MOP`, `GALAHAD_SMT`, and `GALAHAD_NLPT`. **Date:** September 2010. **Origin:** D. P. Robinson, University of Oxford, UK, and N. I. M. Gould, Rutherford Appleton Laboratory. **Language:** Fortran 95 + TR 15581 or Fortran 2003.

2 HOW TO USE THE PACKAGE

The package is available with single, double and (if available) quadruple precision reals, and either 32-bit or 64-bit integers. Access to the 32-bit integer, single precision version requires the `USE` statement

```
USE GALAHAD_CHECK_single
```

with the obvious substitution `GALAHAD_CHECK_double`, `GALAHAD_CHECK_quadruple`, `GALAHAD_CHECK_single_64`, `GALAHAD_CHECK_double_64` and `GALAHAD_CHECK_quadruple_64` for the other variants.

If it is required to use more than one of the modules at the same time, the derived types `SMT_type`, `NLPT_problem_type`, `CHECK_control_type`, `CHECK_inform_type`, `CHECK_data_type`, `CHECK_reverse_communication_type` and `GALAHAD_userdata_type` (Section 2.3), and the subroutines `CHECK_initialize`, `CHECK_verify`, `CHECK_terminate` (Section 2.4), and `CHECK_read_specfile` (Section 2.8) must be renamed on one of the `USE` statements.

2.1 Matrix storage formats

The Jacobian matrix $\mathbf{J} = \nabla_x\mathbf{c}(\mathbf{x})$ and the Hessian matrix $\mathbf{H} = \nabla_{xx}L(\mathbf{x},\mathbf{y})$ may be stored in a variety of input formats.

All use is subject to the conditions of a BSD-3-Clause License.
See `http://galahad.rl.ac.uk/galahad-www/cou.html` for full details.

2.1.1 Dense storage format

The matrix **J** is stored as a compact dense matrix by rows, that is, the values of the entries of each row in turn are stored in order within an appropriate real one-dimensional array. Component $n*(i-1)+j$ of the storage array `J%val` will hold the value j_{ij} for $i = 1,\ldots,m$ and $j = 1,\ldots,n$. Since **H** is symmetric, only the lower triangular part (that is the part h_{ij} for $1 \leq j \leq i \leq n$) should be stored. In this case the lower triangle will be stored by rows, that is component $i*(i-1)/2+j$ of the storage array `H%val` will hold the value h_{ij} (and, by symmetry, h_{ji}) for $1 \leq j \leq i \leq n$.

2.1.2 Sparse co-ordinate storage format

Only the nonzero entries of the matrices are stored. For the l-th entry of **J**, its row index i, column index j and value j_{ij} are stored in the l-th components of the integer arrays `J%row`, `J%col` and real array `J%val`. The order is unimportant, but the total number of entries `J%ne` is also required. Since **H** is symmetric, the same scheme is applicable, except that only the entries in the lower triangle should be stored.

2.1.3 Sparse row-wise storage format

Again only the nonzero entries are stored, but this time they are ordered so that those in row i appear directly before those in row $i+1$. For the i-th row of **J**, the i-th component of a integer array `J%ptr` holds the position of the first entry in this row, while `J%ptr` $(m+1)$ holds the total number of entries plus one. The column indices j and values j_{ij} of the entries in the i-th row are stored in components $l =$ `J%ptr`(i), ...,`J%ptr` $(i+1)-1$ of the integer array `J%col`, and real array `J%val`, respectively. Since **H** is symmetric, the same scheme is applicable, except that only the entries in the lower triangle should be stored.

For sparse matrices, this scheme almost always requires less storage than its predecessor.

2.1.4 Sparse column-wise storage format

Again only the nonzero entries are stored, but this time they are ordered so that those in column j appear directly before those in column $j+1$. For the j-th column of **J**, the j-th component of the integer array `J%ptr` holds the position of the first entry in this column, while `J%ptr` $(n+1)$ holds the total number of entries plus one. The row indices i and values j_{ij} of the entries in the j-th column are stored in components $l =$ `J%ptr`(j), ...,`J%ptr` $(j+1)-1$ of the integer array `J%row`, and real array `J%val`, respectively. Since **H** is symmetric, the same scheme is applicable, except that only the entries in the lower triangle should be stored.

2.1.5 Diagonal storage format

If **J** is diagonal (i.e., $j_{ij} = 0$ for all $1 \leq i \neq j \leq n$) only the diagonal entries j_{ii} for $1 \leq i \leq n$ should be stored, and the first n components of the array `J%val` should be used for this purpose. The same holds for **H**.

2.2 Real and integer kinds

We use the terms integer and real to refer to the fortran keywords `REAL(rp_)` and `INTEGER(ip_)`, where `rp_` and `ip_` are the relevant kind values for the real and integer types employed by the particular module in use. The former are equivalent to default `REAL` for the single precision versions, `DOUBLE PRECISION` for the double precision cases and quadruple-precision if 128-bit reals are available, and correspond to `rp_ = real32`, `rp_ = real64` and `rp_ = real128` respectively as defined by the fortran `iso_fortran_env` module. The latter are default (32-bit) and long (64-bit) integers, and correspond to `ip_ = int32` and `ip_ = int64`, respectively, again from the `iso_fortran_env` module.

All use is subject to the conditions of a BSD-3-Clause License.
See `http://galahad.rl.ac.uk/galahad-www/cou.html` **for full details.**

2.3 The derived data types

Seven derived data types are accessible from the package.

2.3.1 The derived data type for holding matrices

The derived data type SMT_TYPE is used to hold the Jacobian matrix **J** and the Hessian matrix **H**. The components of SMT_TYPE used here are:

m is a scalar component of type INTEGER(ip_) that holds the number of rows of the matrix.

n is a scalar component of type INTEGER(ip_) that holds the number of columns of the matrix.

ne is a scalar variable of type INTEGER(ip_) that holds the number of matrix entries.

type is a rank-one allocatable array of type default CHARACTER that is used to indicate the matrix storage scheme used. Its precise length and content depends on the type of matrix to be stored (see §2.3.2).

val is a rank-one allocatable array of type REAL(rp_) and dimension at least ne, that holds the values of the entries. Each pair of off-diagonal entries $h_{ij} = h_{ji}$ for the *symmetric* matrix **H** is represented as a single entry (see §2.1.1–2.1.3). Any duplicated entries that appear in the sparse co-ordinate, row-wise, or column-wise schemes will be summed.

row is a rank-one allocatable array of type INTEGER(ip_), and dimension at least ne, that may hold the row indices of the entries (see §2.1.2).

col is a rank-one allocatable array of type INTEGER(ip_), and dimension at least ne, that may hold the column indices of the entries (see §2.1.2–2.1.3).

ptr is a rank-one allocatable array of type INTEGER(ip_). If sparse row-wise storage is used (see §2.1.3), then it must have dimension at least m + 1 and hold the pointers to the first entry in each row. If sparse column-wise storage is used (see §2.1.4), then it must have dimension at least n + 1 and hold the pointers to the first entry in each column.

2.3.2 The derived data type for holding the problem

The derived data type NLPT_problem_type holds the problem. The relevant components of NLPT_problem_type are:

m is a scalar variable of type INTEGER(ip_) that holds the number of nonlinear constraints m.

n is a scalar variable of type INTEGER(ip_) that holds the number of optimization variables n.

H is scalar variable of type SMT_TYPE that holds the Hessian matrix $\mathbf{H} = \nabla_{xx} f(\mathbf{x})$. The following components are used here:

H%type is an allocatable array of rank one and type default CHARACTER, that is used to indicate the storage scheme used. If the dense storage scheme (see Section 2.1.1) is used, the first five components of H%type must contain the string DENSE. For the sparse co-ordinate scheme (see Section 2.1.2), the first ten components of H%type must contain the string COORDINATE, for the sparse row-wise storage scheme (see Section 2.1.3), the first fourteen components of H%type must contain the string SPARSE_BY_ROWS, for the sparse column-wise storage scheme (see Section 2.1.4), the first seventeen components of H%type must contain the string SPARSE_BY_COLUMNS, and for the diagonal storage scheme (see Section 2.1.5), the first eight components of H%type must contain the string DIAGONAL.

For convenience, the procedure SMT_put may be used to allocate sufficient space and insert the required keyword into H%type. For example, if nlp is of derived type CHECK_problem_type and involves a Hessian we wish to store using the co-ordinate scheme, we may simply

All use is subject to the conditions of a BSD-3-Clause License.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

```
CALL SMT_put( nlp%H%type, 'COORDINATE' )
```

See the documentation for the GALAHAD package SMT for further details on the use of SMT_put.

H%ne is a scalar variable of type INTEGER(ip_), that holds the number of entries in the **lower triangular** part of **H** in the sparse co-ordinate storage scheme (see Section 2.1.2). It need not be set for any of the other four schemes.

H%val is a rank-one allocatable array of type REAL(rp_), that holds the values of the entries of the **lower triangular** part of the Hessian matrix **H** in any of the storage schemes discussed in Section 2.1.

H%row is a rank-one allocatable array of type INTEGER(ip_), that holds the row indices of the **lower triangular** part of **H** in the sparse co-ordinate storage scheme (see Section 2.1.2) and the sparse column-wise scheme (see Section 2.1.4). It need not be allocated for any of the other three schemes.

H%col is a rank-one allocatable array variable of type INTEGER(ip_), that holds the column indices of the **lower triangular** part of **H** in either the sparse co-ordinate (see Section 2.1.2), or the sparse row-wise (see Section 2.1.3) storage scheme. It need not be allocated for any of the other three schemes.

H%ptr is a rank-one allocatable array of dimension n+1 and type INTEGER(ip_) that holds the starting position of each row of the **lower triangular** part of **H**, as well as the total number of entries plus one, in the sparse row-wise storage scheme (see Section 2.1.3), or the starting position of each column of the **lower triangular** part of **H**, as well as the total number of entries plus one, in the sparse column-wise storage scheme (see Section 2.1.4). It need not be allocated when the other schemes are used.

J is scalar variable of type SMT_TYPE that holds the Jacobian matrix $\mathbf{J} = \nabla_x \mathbf{c}(\mathbf{x})$. The following components are used here:

J%type is an allocatable array of rank one and type default CHARACTER, that is used to indicate the storage scheme used. If the dense storage scheme (see Section 2.1.1) is used, the first five components of J%type must contain the string DENSE. For the sparse co-ordinate scheme (see Section 2.1.2), the first ten components of J%type must contain the string COORDINATE, for the sparse row-wise storage scheme (see Section 2.1.3), the first fourteen components of J%type must contain the string SPARSE_BY_ROWS, for the sparse column-wise storage scheme (see Section 2.1.4), the first seventeen components of J%type must contain the string SPARSE_BY_COLUMNS, and for the diagonal storage scheme (see Section 2.1.5), the first eight components of J%type must contain the string DIAGONAL.

For convenience, the procedure SMT_put may be used to allocate sufficient space and insert the required keyword into J%type. For example, if nlp is of derived type CHECK_problem_type and involves a Jacobian we wish to store using the co-ordinate scheme, we may simply

```
CALL SMT_put( nlp%J%type, 'COORDINATE' )
```

See the documentation for the GALAHAD package SMT for further details on the use of SMT_put.

J%ne is a scalar variable of type INTEGER(ip_) that holds the number of entries in the sparse co-ordinate storage scheme (see Section 2.1.2). It need not be set for any of the other schemes.

J%val is a rank-one allocatable array of type REAL(rp_) that holds the values of the entries of the Jacobian matrix **J** in any of the storage schemes discussed in Section 2.1.

J%row is a rank-one allocatable array of type INTEGER(ip_) that holds the row indices of **J** in the sparse co-ordinate storage scheme discussed in Section 2.1.2 and the sparse column-wise storage scheme discussed in Section 2.1.4. It need not be allocated for any of the other three schemes.

J%col is a rank-one allocatable array variable of type INTEGER(ip_) that holds the column indices of **J** in either the sparse co-ordinate scheme discussed in Section 2.1.2 or the sparse row-wise scheme discussed in Section 2.1.3. It need not be allocated for any of the other three schemes.

All use is subject to the conditions of a BSD-3-Clause License.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

J%ptr for the sparse row-wise storage scheme discussed in Section 2.1.3, it is a rank-one allocatable array of dimension m+1 and type INTEGER(ip_) that holds the starting position of each row of **J** as well as the total number of entries plus one. For the sparse column-wise storage scheme discussed in Section 2.1.4, it is a rank-one allocatable array of dimension n+1 and type INTEGER(ip_) that holds the starting position of each column of **J** as well as the total number of entries plus one. It need not be allocated for any of the other schemes.

G is a rank-one allocatable array of dimension n and type REAL(rp_), that holds the gradient **g** of the objective function. The j-th component of G, $j = 1,\ldots,n$, contains $\mathbf{g}_j$.

C is a rank-one allocatable array of dimension m and type REAL(rp_) that holds the value of the constraint function. The j-th component of C, $j = 1,\ldots,m$, contains $\mathbf{c}_j$.

f is a scalar variable of type REAL(rp_) that holds the value of the objective function.

X is a rank-one allocatable array of dimension n and type REAL(rp_), that holds the values **x** of the optimization variables. The j-th component of X, $j = 1,\ldots,n$, contains x_j.

X_l is a rank-one allocatable array of dimension n and type REAL(rp_) that holds the lower bounds on the optimization variables **x**.

X_u is a rank-one allocatable array of dimension n and type REAL(rp_) that holds upper bounds on the optimization variables **x**.

Y is a rank-one allocatable array of dimension m and type REAL(rp_) that holds the value **y** of the Lagrange multiplier estimate. The j-th component of Y, $j = 1,\ldots,m$, contains y_j.

2.3.3 The derived data type for holding control parameters

The derived data type CHECK_control_type is used to hold controlling data. Default values may be obtained by calling CHECK_initialize (see Section 2.4.1), while components may also be changed by calling GALAHAD_CHECK_read_spec (see Section 2.8.1). The components of CHECK_control_type are:

error is a scalar variable of type INTEGER(ip_), that holds the stream number for error messages. Printing of error messages in CHECK_verify and CHECK_terminate is suppressed if error ≤ 0. The default is error = 6.

out is a scalar variable of type INTEGER(ip_), that holds the stream number for informational messages. Printing of informational messages in CHECK_verify is suppressed if out ≤ 0. The default is out = 6.

print_level is a scalar variable of type INTEGER(ip_), that is used to control the amount of informational output which is required. No informational output will occur if print_level ≤ 0. If print_level $= 1$, a brief summary of the derivative verification is provided. If print_level ≥ 2, this output will be increased to provide significant detail of each iteration (see Section 2.9 for more details). The default is print_level = 0.

verify_level is a scalar variable of type INTEGER(ip_) that determines the detail of verification performed. A "cheap" check will be performed if verify_level $= 1$. If verify_level ≥ 2, an "expensive"—but more detailed—verification of the derivatives is done. No checking is performed if verify_level ≤ 0. The default is verify_level = 2.

f_availability is a scalar variable of type INTEGER(ip_) that controls how the user is expected to supply objective function values, when required. The user should set f_availability $= 1$ if an appropriate evaluation routine is supplied (see Section 2.5.1), and f_availability $= 2$ if reverse communication will be used (see Section 2.6).

All use is subject to the conditions of a BSD-3-Clause License.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

c_availability is a scalar variable of type INTEGER(ip_) that controls how the user is expected to supply constraint function values, when required. The user should set c_availability = 1 if an appropriate evaluation routine is supplied (see Section 2.5.2), and c_availability = 2 if reverse communication will be used (see Section 2.6). Any other value will result in an error message.

g_availability is a scalar variable of type INTEGER(ip_) that controls how the user is expected to supply the gradient of the objective function, when required. The user should set g_availability = 1 if an appropriate evaluation routine is supplied (see Section 2.5.3), and g_availability = 2 if reverse communication will be used (see Section 2.6). Any other value will result in an error message.

J_availability is a scalar variable of type INTEGER(ip_) that controls how the user is expected to supply the Jacobian of the constraint function, when required. The user should set J_availability = 1 if an appropriate evaluation routine is supplied (see Section 2.5.4), J_availability = 2 if reverse communication will be used to obtain Jacobian values (see Section 2.6), J_availability = 3 if an appropriate Jacobian-vector product routine is supplied (see Section 2.5.6), and J_availability = 4 if reverse communication will be used to get Jacobian-vector products (see Section 2.6). Any other value will result in an error message.

H_availability is a scalar variable of type INTEGER(ip_) that controls how the user is expected to supply the Hessian of the Lagrangian function, when required. The user should set H_availability = 1 if an appropriate evaluation routine is supplied (see Section 2.5.5), H_availability = 2 if reverse communication will be used to obtain Hessian values (see Section 2.6), H_availability = 3 if an appropriate Hessian-vector product routine is supplied (see Section 2.5.7), and H_availability = 4 if reverse communication will be used to get Hessian-vector products (see Section 2.6). Any other value will result in an error message.

checkG is a scalar variable of type default LOGICAL that should be set .TRUE. if the gradient of the objective function should be checked. Otherwise, it should be set .FALSE.. The default is checkG=.TRUE..

checkJ is a scalar variable of type default LOGICAL that should be set .TRUE. if the Jacobian of the constraint function should be checked. Otherwise, it should be set .FALSE.. The default is checkJ =.TRUE..

checkH is a scalar variable of type default LOGICAL that should be set .TRUE. if the Hessian of the Lagrangian function should be checked. Otherwise, it should be set .FALSE.. The default is checkH=.TRUE..

deallocate_error_fatal is a scalar variable of type default LOGICAL, that must be set .TRUE. if the user wishes to terminate execution if a deallocation fails, and .FALSE. if an attempt to continue will be made. The default is deallocate_error_fatal = .FALSE..

2.3.4 The derived data type for holding informational parameters

The derived data type CHECK_inform_type is used to hold parameters that give information about the progress and needs of the algorithm. The components of CHECK_inform_type are:

status is a scalar variable of type INTEGER(ip_) that gives the exit status of the algorithm. See Sections 2.6 and 2.7 for further details.

alloc_status is a scalar variable of type INTEGER(ip_) that gives the status of the last attempted array allocation or deallocation. This will be 0 if status = 0.

bad_alloc is a scalar variable of type default CHARACTER and length 80 that gives the name of the last internal array for which there was an allocation or deallocation error. This will be the null string if status = 0.

numG_wrong is a scalar variable of type INTEGER(ip_) that gives the number of components of the gradient of the objective function that appear to be wrong.

All use is subject to the conditions of a BSD-3-Clause License.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

numJ_wrong is a scalar variable of type INTEGER(ip_) that gives the number of entries of the Jacobian of the constraint function that appear to be wrong.

numH_wrong is a scalar variable of type INTEGER(ip_) that gives the number of entries of the Hessian of the Lagrangian function that appear to be wrong.

derivative_ok is a scalar variable of type default LOGICAL that is .TRUE. if all derivatives appear to be correct, and set .FALSE. otherwise.

2.3.5 The derived data type for holding problem data

The derived data type CHECK_data_type is used to hold all the data for a particular problem, or sequences of problems with the same structure, between calls of CHECK procedures. The only data that should be changed by the user from the initial call to CHECK_initialize to the final call to CHECK_terminate is the component RC of type CHECK_reverse_communication_type (see Section 2.3.6), and this should be done only as directed from GALAHAD_CHECK_verify with positive values of inform%status as described in Section 2.6.

2.3.6 The derived data type for holding reverse communication data

The derived data type CHECK_reverse_communication_type is used to hold data for reverse communication, when needed. The components of CHECK_reverse_communication_type are:

X is a rank-one allocatable array of type REAL(rp_) that holds the values of the optimization variables at which the user must perform function computation.

Y is a rank-one allocatable array of type REAL(rp_) that holds the values of the Lagrange multipliers that the user must use when evaluating the Hessian of the Lagrangian.

F is a scalar variable of type REAL(rp_) in which the user places the value of the objective function evaluated at X, when required (see Section 2.6).

C is a rank-one allocatable array of type REAL(rp_) in which the user places the value of the constraint function evaluated at X, when required (see Section 2.6).

G is a rank-one allocatable array of type REAL(rp_) in which the user places the gradient of the the objective function evaluated at X, when required (see Section 2.6).

V is a rank-one allocatable array of type REAL(rp_) that holds the vector for which a matrix-vector product is required (see Section 2.6).

U is a rank-one allocatable array of type REAL(rp_) in which the user places the result of any required matrix-vector product with the vector V from above (see Section 2.6).

Jval is a rank-one allocatable array of type REAL(rp_) in which the user places the entries of the Jacobian matrix evaluated at X, when required (see Section 2.6).

Hval is a rank-one allocatable array of type REAL(rp_) in which the user places the **lower triangular** entries of the Hessian matrix of the Lagrangian evaluated at X and Y, when required (see Section 2.6).

All use is subject to the conditions of a BSD-3-Clause License.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

2.3.7 The derived data type for holding user data

The derived data type GALAHAD_userdata_type is available from the package GALAHAD_userdata to allow the user to pass data to and from user-supplied subroutines for function and derivative calculations (see Section 2.5). Components of variables of type GALAHAD_userdata_type may be allocated as necessary. The following components are available:

integer is a rank-one allocatable array of type INTEGER(ip_).

real is a rank-one allocatable array of type default REAL(rp_)

complex is a rank-one allocatable array of type default COMPLEX (double precision complex in GALAHAD_CHECK_double).

character is a rank-one allocatable array of type default CHARACTER.

logical is a rank-one allocatable array of type default LOGICAL.

integer_pointer is a rank-one pointer array of type INTEGER(ip_).

real_pointer is a rank-one pointer array of type default REAL(rp_)

complex_pointer is a rank-one pointer array of type default COMPLEX (double precision complex in GALAHAD_CHECK_double).

character_pointer is a rank-one pointer array of type default CHARACTER.

logical_pointer is a rank-one pointer array of type default LOGICAL.

2.4 Argument lists and calling sequences

There are three procedures for user calls (see Section 2.8 for further features):

1. The subroutine CHECK_initialize is used to set default values, and initialize private data, before verifying the derivatives of one or more problems with the same sparsity and bound structure.
2. The subroutine CHECK_verify is called to check the derivatives of the given problem.
3. The subroutine CHECK_terminate is provided to allow the user to automatically deallocate array components of the private data, allocated by CHECK_verify, at the end of the verification process. It is important to do this if the data object is re-used for another problem **with a different structure** since CHECK_initialize cannot test for this situation, and any existing associated targets will subsequently become unreachable.

We use square brackets [ ] to indicate OPTIONAL arguments.

2.4.1 The initialization subroutine

Default values are provided as follows:

```
CALL CHECK_initialize( control )
```

control is a scalar INTENT(OUT) argument of type CHECK_control_type (see Section 2.3.3). On exit, control contains default values for the components as described in Section 2.3.3. These values should only be changed after calling CHECK_initialize.

All use is subject to the conditions of a BSD-3-Clause License.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

2.4.2 The verification subroutine

The verification algorithm is called as follows:

```
CALL CHECK_verify( nlp, data, control, inform, userdata[, eval_F, eval_C,     &
                   eval_G, eval_J, eval_H, eval_Jv, eval_Hv] )
```

`nlp` is a scalar `INTENT(INOUT)` argument of type `NLPT_problem_type` (see Section 2.3.2). It is used to hold data about the problem whose derivatives are being verified. For a new problem, the user must allocate all the array components, and set values for `nlp%m`, `nlp%n`, and the required integer components of `nlp%J` and `nlp%H` that is determined by the values of `checkJ` and `checkH` as described in Section 2.3.3. Users are free to choose whichever of the matrix formats described in Section 2.1 is appropriate for $\mathbf{J}$ and $\mathbf{H}$ for their application.

The components `nlp%X` and `nlp%Y` must be set to initial values $\mathbf{x}$ and $\mathbf{y}$ of the primal and dual variables for the optimization problem. Prior to verification of the derivatives, the point `nlp%X` is modified internally to ensure feasibility with respect to the bound constraints $\mathbf{x}^l$ and $\mathbf{x}^u$; no modification of $\mathbf{y}$ is performed. The requested derivatives are then checked at the point `nlp%X` and `nlp%Y`.

Restrictions: `nlp%n` > 0 and `nlp%m` ≥ 0.

`data` is a scalar `INTENT(INOUT)` argument of type `CHECK_data_type` (see Section 2.3.5). It is used to hold data about the problem derivatives being verified. With the possible exception of the component `RC` (see Sections 2.3.6 and 2.6), it must not have been altered **by the user** since the last call to `CHECK_initialize`.

`control` is a scalar `INTENT(IN)` argument of type `CHECK_control_type` (see Section 2.3.3). Default values may be assigned by calling `CHECK_initialize` prior to the first call to `CHECK_solve`.

`inform` is a scalar `INTENT(INOUT)` argument of type `CHECK_inform_type` (see Section 2.3.4). **On initial entry, the component** `status` **must be set to the value 1.** Other entries need not be set. A successful call to `CHECK_verify` is indicated when the component `status` has the value 0. For other return values of `status`, see Sections 2.6 and 2.7.

`userdata` is a scalar `INTENT(INOUT)` argument of type `GALAHAD_userdata_type` whose components may be used to communicate user-supplied data to and from the `OPTIONAL` subroutines `eval_F`, `eval_C`, `eval_G`, `eval_J`, `eval_H`, `eval_Jv`, and `eval_Hv` (see Section 2.3.7).

`eval_F` is an `OPTIONAL` user-supplied subroutine whose purpose is to evaluate the objective function $f(\mathbf{x})$ at a given vector $\mathbf{x}$. See Section 2.5.1 for details. If `f_availability` $= 1$ (see Section 2.3.3), then `eval_F` must be present and declared `EXTERNAL` in the calling program. If `f_availability` $= 2$, then `GALAHAD_CHECK_verify` will use reverse communication to obtain objective function values (see Section 2.6).

`eval_C` is an `OPTIONAL` user-supplied subroutine whose purpose is to evaluate the constraint function $\mathbf{c}(\mathbf{x})$ at a given vector $\mathbf{x}$. See Section 2.5.2 for details. If `c_availability` $= 1$ (see Section 2.3.3), then `eval_C` must be present and declared `EXTERNAL` in the calling program. If `c_availability` $= 2$, then `GALAHAD_CHECK_verify` will use reverse communication to obtain constraint function values (see Section 2.6).

`eval_G` is an `OPTIONAL` user-supplied subroutine whose purpose is to evaluate the gradient of the objective function $\nabla_x f(\mathbf{x})$ at a given vector $\mathbf{x}$. See Section 2.5.3 for details. If `g_availability` $= 1$ (see Section 2.3.3), then `eval_G` must be present and declared `EXTERNAL` in the calling program. If `g_availability` $= 2$, then `GALAHAD_CHECK_verify` will use reverse communication to obtain gradient values (see Section 2.6).

`eval_J` is an `OPTIONAL` user-supplied subroutine whose purpose is to evaluate the Jacobian of the objective function $\nabla_{xx} f(\mathbf{x})$ at a given vector $\mathbf{x}$. See Section 2.5.5 for details. If `J_availability` $= 1$ (see Section 2.3.3), then `eval_J` must be present and declared `EXTERNAL` in the calling program. Otherwise, `eval_J` need not be supplied.

`eval_H` is an `OPTIONAL` user-supplied subroutine whose purpose is to evaluate the Hessian of the Lagrangian $\nabla_{xx} L(\mathbf{x},\mathbf{y})$ at a given point $(\mathbf{x},\mathbf{y})$. See Section 2.5.5 for details. If `H_availability` $= 1$ (see Section 2.3.3), then `eval_H` must be present and declared `EXTERNAL` in the calling program. Otherwise, `eval_H` need not be supplied.

All use is subject to the conditions of a BSD-3-Clause License.
See `http://galahad.rl.ac.uk/galahad-www/cou.html` **for full details.**

eval_Jv is an OPTIONAL user-supplied subroutine whose purpose is to evaluate the value of the product of the Jacobian of the constraint function $\nabla_x \mathbf{c}(\mathbf{x})$ with a given vector $\mathbf{v}$. See Section 2.5.7 for details. If J_availability $= 3$ (see Section 2.3.3), then eval_Jv must be present and declared EXTERNAL in the calling program. Otherwise, eval_Jv need not be supplied.

eval_Hv is an OPTIONAL user-supplied subroutine whose purpose is to evaluate the value of the product of the Hessian of the Lagrangian function $\nabla_{xx} L(\mathbf{x}, \mathbf{y})$ with a given vector $\mathbf{v}$. See Section 2.5.7 for details. If H_availability $= 3$ (see Section 2.3.3), then eval_Hv must be present and declared EXTERNAL in the calling program. Otherwise, eval_Hv need not be supplied.

2.4.3 The termination subroutine

All previously allocated arrays are deallocated as follows:

```
CALL CHECK_terminate( data, control, inform )
```

data is a scalar INTENT(INOUT) argument of type CHECK_data_type exactly as for CHECK_verify, which must not have been altered **by the user** since the last call to CHECK_initialize (except possibly data%RC as described in Section 2.6). On exit, array components will have been deallocated.

control is a scalar INTENT(IN) argument of type CHECK_control_type exactly as for CHECK_verify.

inform is a scalar INTENT(OUT) argument of type CHECK_inform_type exactly as for CHECK_verify. Only the component status will be set on exit, and a successful call to CHECK_terminate is indicated when this component status has the value 0. For other return values of status see Section 2.7.

2.5 Function and derivative values

2.5.1 The objective function value via internal evaluation

If the control parameter f_availability $= 1$ (see Section 2.3.3), then the argument eval_F must be present when calling GALAHAD_CHECK_verify and the user must provide a subroutine of that name to evaluate the value of the objective function $f(\mathbf{x})$. The routine must be specified as

```
SUBROUTINE eval_F( status, X, userdata, F )
```

whose arguments are as follows:

status is a scalar INTENT(OUT) argument of type INTEGER(ip_), that should be set to 0 if the routine has been able to evaluate the objective function and to a non-zero value if the evaluation has not been possible.

X is a rank-one INTENT(IN) array argument of type REAL(rp_) whose components contain the vector $\mathbf{x}$.

userdata is a scalar INTENT(INOUT) argument of type GALAHAD_userdata_type whose components may be used to communicate user-supplied data to and from the subroutines eval_F, eval_C, eval_G, eval_J eval_H, eval_Jv, and eval_Hv (see Section 2.3.7).

F is a scalar INTENT(OUT) argument of type REAL(rp_) that should be set to the value of the objective function $f(\mathbf{x})$ evaluated at the vector $\mathbf{x}$ input in X.

All use is subject to the conditions of a BSD-3-Clause License.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

2.5.2 The constraint function value via internal evaluation

If the control parameter c_availability $= 1$ (see Section 2.3.3), then the argument eval_C must be present when calling GALAHAD_CHECK_verify and the user must provide a subroutine of that name to evaluate the value of the constraint function $\mathbf{c}(\mathbf{x})$. The routine must be specified as

```
SUBROUTINE eval_C( status, X, userdata, C )
```

whose arguments are as follows:

status is a scalar INTENT(OUT) argument of type INTEGER(ip_), that should be set to 0 if the routine has been able to evaluate the constraint function and to a non-zero value if the evaluation has not been possible.

X is a rank-one INTENT(IN) array argument of type REAL(rp_) whose components contain the vector $\mathbf{x}$.

userdata is a scalar INTENT(INOUT) argument of type GALAHAD_userdata_type whose components may be used to communicate user-supplied data to and from the subroutines eval_F, eval_C, eval_G, eval_J eval_H, eval_Jv, and eval_Hv (see Section 2.3.7).

C is a rank-one INTENT(OUT) argument of type REAL(rp_) that should be set to the value of the constraint function $\mathbf{c}(\mathbf{x})$ evaluated at the vector $\mathbf{x}$ input in X.

2.5.3 Gradient values via internal evaluation

If the control parameter g_availability $= 1$ (see Section 2.3.3), then the argument eval_G must be present when calling GALAHAD_CHECK_verify and the user must provide a subroutine of that name to evaluate the value of the gradient the objective function $\nabla_x f(\mathbf{x})$. The routine must be specified as

```
SUBROUTINE eval_G( status, X, userdata, G )
```

whose arguments are as follows:

status is a scalar INTENT(OUT) argument of type INTEGER(ip_) that should be set to 0 if the routine has been able to evaluate the gradient of the objective function and to a non-zero value if the evaluation has not been possible.

X is a rank-one INTENT(IN) array argument of type REAL(rp_) whose components contain the vector $\mathbf{x}$.

userdata is a scalar INTENT(INOUT) argument of type GALAHAD_userdata_type whose components may be used to communicate user-supplied data to and from the subroutines eval_F, eval_C, eval_G, eval_J eval_H, eval_Jv, and eval_Hv (see Section 2.3.7).

G is a rank-one INTENT(OUT) argument of type REAL(rp_), whose components should be set to the values of the gradient of the objective function $\nabla_x f(\mathbf{x})$ evaluated at the vector $\mathbf{x}$ input in X.

2.5.4 Jacobian values via internal evaluation

If the control parameter J_availability $= 1$ (see Section 2.3.3), then the argument eval_J must be present when calling GALAHAD_CHECK_verify and the user must provide a subroutine of that name to evaluate the values of the Jacobian of the constraint function $\nabla_x \mathbf{c}(\mathbf{x})$. The routine must be specified as

```
SUBROUTINE eval_J( status, X, userdata, Jval )
```

whose arguments are as follows:

All use is subject to the conditions of a BSD-3-Clause License.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

status is a scalar INTENT(OUT) argument of type INTEGER(ip_) that should be set to 0 if the routine has been able to evaluate the Jacobian of the constraint function and to a non-zero value if the evaluation has not been possible.

X is a rank-one INTENT(IN) array argument of type REAL(rp_) whose components contain the vector $\mathbf{x}$.

userdata is a scalar INTENT(INOUT) argument of type GALAHAD_userdata_type whose components may be used to communicate user-supplied data to and from the subroutines eval_F, eval_C, eval_G, eval_J eval_H, eval_Jv, and eval_Hv (see Section 2.3.7).

Jval is a scalar INTENT(OUT) argument of type REAL(rp_), whose components should be set to the values of the Jacobian of the constraint function $\nabla_x \mathbf{c}(\mathbf{x})$ evaluated at the vector $\mathbf{x}$ input in X. The values should be input in the same order as that in which the array indices were given in nlp%J.

2.5.5 Hessian values via internal evaluation

If the control parameter H_availability $= 1$ (see Section 2.3.3), then the argument eval_H must be present when calling GALAHAD_CHECK_verify and the user must provide a subroutine of that name to evaluate the values of the Hessian of the Lagrangian $\nabla_{xx} L(\mathbf{x}, \mathbf{y})$. The routine must be specified as

```
SUBROUTINE eval_H( status, X, Y, userdata, Hval )
```

whose arguments are as follows:

status is a scalar INTENT(OUT) argument of type INTEGER(ip_), that should be set to 0 if the routine has been able to evaluate the Hessian of the Lagrangian and to a non-zero value if the evaluation has not been possible.

X is a rank-one INTENT(IN) array argument of type REAL(rp_) whose components contain the vector $\mathbf{x}$.

Y is a rank-one INTENT(IN) array argument of type REAL(rp_) whose components contain the dual vector $\mathbf{y}$.

userdata is a scalar INTENT(INOUT) argument of type GALAHAD_userdata_type whose components may be used to communicate user-supplied data to and from the subroutines eval_F, eval_C, eval_G, eval_J eval_H, eval_Jv, and eval_Hv (see Section 2.3.7).

Hval is a scalar INTENT(OUT) argument of type REAL(rp_), whose components should be set to the values of the Hessian of the Lagrangian $\nabla_{xx} L(\mathbf{x}, \mathbf{y})$ evaluated at the vector $(\mathbf{x}, \mathbf{y})$ input in X and Y. The values should be input in the same order as that in which the array indices were given in nlp%H.

2.5.6 Jacobian-vector products via internal evaluation

If the control parameter J_availability $= 3$ (see Section 2.3.3), then the argument eval_Jv must be present when calling GALAHAD_CHECK_verify and the user must provide a subroutine of that name to compute products of the Jacobian of the constraint function (and its transpose) of the form $\mathbf{u} + \nabla_x \mathbf{c}(\mathbf{x})\mathbf{v}$ and $\mathbf{u} + \nabla_x \mathbf{c}(\mathbf{x})^T \mathbf{v}$. The routine must be specified as

```
SUBROUTINE eval_Jv( status, X, userdata, transpose, U, V )
```

whose arguments are as follows:

status is a scalar INTENT(OUT) argument of type INTEGER(ip_) that should be set to 0 if the routine has been able to perform the required calculation (see transpose below) and to a non-zero value if the computation has not been possible.

X is a rank-one INTENT(IN) array argument of type REAL(rp_) whose components contain the vector $\mathbf{x}$.

All use is subject to the conditions of a BSD-3-Clause License.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

userdata is a scalar INTENT(INOUT) argument of type GALAHAD_userdata_type whose components may be used to communicate user-supplied data to and from the subroutines eval_F, eval_C, eval_G, eval_J eval_H, eval_Jv, and eval_Hv (see Section 2.3.7).

transpose is a scalar INTENT(IN) argument of type default LOGICAL. If transpose = .FALSE., then the user should evaluate the sum $\mathbf{u} + \nabla_x \mathbf{c}(\mathbf{x})\mathbf{v}$. If transpose = .TRUE., then the user should evaluate the sum $\mathbf{u} + \nabla_x \mathbf{c}(\mathbf{x})^T\mathbf{v}$.

U is a rank-one INTENT(INOUT) array argument of type REAL(rp_) whose components on input contain the vector $\mathbf{u}$ and on output contains either the sum $\mathbf{u} + \nabla_x \mathbf{c}(\mathbf{x})\mathbf{v}$ or $\mathbf{u} + \nabla_x \mathbf{c}(\mathbf{x})^T\mathbf{v}$ depending on the value of transpose given above.

V is a rank-one INTENT(IN) array argument of type REAL(rp_) whose components contain the vector $\mathbf{v}$.

2.5.7 Hessian-vector products via internal evaluation

If the control parameter H_availability = 3 (see Section 2.3.3), then the argument eval_Hv must be present when calling GALAHAD_CHECK_verify and the user must provide a subroutine of that name to evaluate the sum $\mathbf{u} + \nabla_{xx} L(\mathbf{x},\mathbf{y})\mathbf{v}$ involving the product of the Hessian of the Lagrangian $\nabla_{xx} L(\mathbf{x},\mathbf{y})$. The routine must be specified as

```
SUBROUTINE eval_Hv( status, X, Y, userdata, U, V )
```

whose arguments are as follows:

status is a scalar INTENT(OUT) argument of type INTEGER(ip_) that should be set to 0 if the routine has been able to perform the required calculation and to a non-zero value if the computation has not been possible.

X is a rank-one INTENT(IN) array argument of type REAL(rp_) whose components contain the vector $\mathbf{x}$.

Y is a rank-one INTENT(IN) array argument of type REAL(rp_) whose components contain the vector $\mathbf{y}$.

userdata is a scalar INTENT(INOUT) argument of type GALAHAD_userdata_type whose components may be used to communicate user-supplied data to and from the subroutines eval_F, eval_C, eval_G, eval_J eval_H, eval_Jv, and eval_Hv (see Section 2.3.7).

U is a rank-one INTENT(INOUT) array argument of type REAL(rp_) whose components on input contain the vector $\mathbf{u}$ and on output the sum $\mathbf{u} + \nabla_{xx} L(\mathbf{x},\mathbf{y})\mathbf{v}$.

V is a rank-one INTENT(IN) array argument of type REAL(rp_) whose components contain the vector $\mathbf{v}$.

2.6 Reverse Communication Information

A positive value of inform%status on exit from CHECK_verify indicates that GALAHAD_CHECK_verify is seeking further information—this will happen if the user has chosen not to evaluate function or derivative values internally (see Sections 2.3.3 and 2.5). The user should compute the required information and re-enter GALAHAD_CHECK_verify with all arguments (except those specifically mentioned below) unchanged.

Possible values of inform%status and the information required are

2. The user should compute the objective function value $f(\mathbf{x})$ at the point $\mathbf{x}$ indicated in data%RC%X. The required value should be set in data%RC%F. If the user is unable to evaluate $f(\mathbf{x})$—for instance, if the function is undefined at $\mathbf{x}$—the user need not set data%RC%F, but should then set inform%status to any negative value. Otherwise, the value of inform%status should remain unchanged.

All use is subject to the conditions of a BSD-3-Clause License.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

3. The user should compute the constraint function value $\mathbf{c}(\mathbf{x})$ at the point $\mathbf{x}$ indicated in data%RC%X. The required value should be set in data%RC%C. If the user is unable to evaluate $\mathbf{c}(\mathbf{x})$—for instance, if the function is undefined at $\mathbf{x}$—the user need not set data%RC%C, but should then set inform%status to any negative value. Otherwise, the value of inform%status should remain unchanged.

4. The user should compute the gradient of the objective function $\nabla_x f(\mathbf{x})$ at the point $\mathbf{x}$ indicated in data%RC%X. The value of the i-th component of the gradient should be set in data%RC%G(i) for $i = 1,\ldots,n$. If the user is unable to evaluate a component of $\nabla_x f(\mathbf{x})$—for instance, if a component of the gradient is undefined at $\mathbf{x}$—the user need not set data%RC%G, but should then set inform%status to a negative value. Otherwise, the value of inform%status should remain unchanged.

5. The user should compute the Jacobian of the constraint function $\nabla_x \mathbf{c}(\mathbf{x})$ at the point $\mathbf{x}$ indicated in data%RC%X. The l-th component of the Jacobian stored according to the scheme used to input nlp%J (see Section 2.3.2) should be set in data%RC%Jval(l) for $l = 1,\ldots,$ nlp%J%ne. If the user is unable to evaluate a component of $\nabla_x \mathbf{c}(\mathbf{x})$—for instance, if a component of the Jacobian is undefined at $\mathbf{x}$—the user need not set data%RC%Jval, but should then set inform%status to a negative number. Otherwise, the value of inform%status should remain unchanged.

6. The user should compute the value $\mathbf{u} + \nabla_x \mathbf{c}(\mathbf{x})\mathbf{v}$, which requires a product of the Jacobian of the constraint function $\nabla_x \mathbf{c}(\mathbf{x})$ at the point $\mathbf{x}$ with the vector $\mathbf{v}$; the vectors $\mathbf{x}$, $\mathbf{u}$, and $\mathbf{v}$ are contained in data%RC%X, data%RC%U, and data%RC%V, respectively. On exit, the resulting vector $\mathbf{u} + \nabla_x \mathbf{c}(\mathbf{x})\mathbf{v}$ should be stored in data%RC%U. If the user is unable to evaluate the product—for instance, if a component of the Jacobian is undefined at $\mathbf{x}$—the user need not set data%RC%U, but should then set inform%status to a negative value. Otherwise, the value of inform%status should remain unchanged.

7. The user should compute the value $\mathbf{u} + \nabla_x \mathbf{c}(\mathbf{x})^T\mathbf{v}$, which requires a product of the transpose of the Jacobian of the constraint function $\nabla_x \mathbf{c}(\mathbf{x})$ at the point $\mathbf{x}$ with the vector $\mathbf{v}$; the vectors $\mathbf{x}$, $\mathbf{u}$, and $\mathbf{v}$ are contained in data%RC%X, data%RC%U, and data%RC%V, respectively. On exit, the resulting vector $\mathbf{u} + \nabla_x \mathbf{c}(\mathbf{x})^T\mathbf{v}$ should be stored in data%RC%U. If the user is unable to evaluate the product—for instance, if a component of the Jacobian is undefined at $\mathbf{x}$—the user need not set data%RC%U, but should then set inform%status to a negative value. Otherwise, the value of inform%status should remain unchanged.

8. The user should compute the Hessian of the Lagrangian $\nabla_{xx} L(\mathbf{x},\mathbf{y})$ at the point $(\mathbf{x},\mathbf{y})$ indicated in data%RC%X and data%RC%Y. The l-th component of the Hessian stored according to the scheme used to input nlp%H (see Section 2.3.2) should be set in data%RC%Hval(l) for $l = 1,\ldots,$ nlp%H%ne. If the user is unable to evaluate a component of $\nabla_{xx} L(\mathbf{x},\mathbf{y})$—for instance, if a component of the Hessian is undefined at $(\mathbf{x},\mathbf{y})$—the user need not set data%RC%Hval, but should then set inform%status to a negative value. Otherwise, the value of inform%status should remain unchanged.

9. The user should compute the value $\mathbf{u} + \nabla_{xx} L(\mathbf{x},\mathbf{y})\mathbf{v}$, which requires a product of the Hessian of the Lagrangian $\nabla_{xx} L(\mathbf{x},\mathbf{y})$ at the point $(\mathbf{x},\mathbf{y})$ with the vector $\mathbf{v}$; the vectors $\mathbf{x}$, $\mathbf{y}$, $\mathbf{u}$, and $\mathbf{v}$ are contained in data%RC%X, data%RC%Y, data%RC%U, and data%RC%V, respectively. On exit, the resulting vector $\mathbf{u} + \nabla_{xx} L(\mathbf{x},\mathbf{y})\mathbf{v}$ should be stored in data%RC%U. If the user is unable to evaluate the product—for instance, if a component of the Hessian is undefined at $(\mathbf{x},\mathbf{y})$—the user need not set data%RC%U, but should then set inform%status to a negative value. Otherwise, the value of inform%status should remain unchanged.

2.7 Warning and error messages

A negative value of inform%status on exit from CHECK_verify or CHECK_terminate indicates that an error has occurred. No further calls should be made until the error has been corrected. Possible values are:

-1. An allocation error occurred. A message indicating the offending array is written on unit control%error, and the returned allocation status and a string containing the name of the offending array are held in inform%alloc_status and inform%bad_alloc, respectively.

All use is subject to the conditions of a BSD-3-Clause License.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

-2. A deallocation error occurred. A message indicating the offending array is written on unit control%error and the returned allocation status and a string containing the name of the offending array are held in inform%alloc_status and inform%bad_alloc, respectively.

-3. Either one of the restrictions nlp%n >0 or nlp%m ≥ 0 is violated, or the requirement that nlp%J_type and nlp%H_type contain a relevant string 'DENSE', 'COORDINATE', 'SPARSE_BY_ROWS', 'SPARSE_BY_COLUMNS', or 'DIAGONAL' is not satisfied.

-50. The user has called CHECK_verify with inform%status <0, which indicates that the user was not able to perform a requested computation needed during reverse communication.

-51. The user has called CHECK_verify with inform%status $=0$, which should never happen. The user should only set inform%status in two situations: inform%status $=1$ prior to the initial call to CHECK_verify, and inform%status <0 when reverse communication is being used and the user is unable to perform the required computation as indicated by the value of inform%status on return from CHECK_verify (see Section 2.6). The user should not change inform%status for any other reason.

-55. The user has input an invalid value for at least one of the control parameters f_availability, c_availability, g_availability, J_availability, or H_availability as described in Section 2.3.3.

-56. Based on the values of the control parameters f_availability, c_availability, g_availability, J_availability, and H_availability (see Section 2.3.3), at least one optional dummy subroutine is missing in the call to CHECK_verify.

-57. At least one component of nlp%X_l or nlp%X_u is inappropriate (see Section 2.3.2).

-58. A user supplied function (see Sections 2.5.1–2.5.7) returned inform%status $\neq 0$, implying that the function computation could not be performed at the required point.

2.8 Further features

In this section, we describe an alternative means of setting control parameters—that is components of the variable control of type CHECK_control_type (see Section 2.3.3)—by reading an appropriate data specification file using the subroutine CHECK_read_specfile. This facility is useful as it allows a user to change CHECK control parameters without editing and recompiling programs that call CHECK.

A specification file, or specfile, is a data file containing a number of "specification commands". Each command occurs on a separate line, and comprises a "keyword", which is a string (in a close-to-natural language) used to identify a control parameter, and an (optional) "value", which defines the value to be assigned to the given control parameter. All keywords and values are case insensitive, keywords may be preceded by one or more blanks but values must not contain blanks, and each value must be separated from its keyword by at least one blank. Values must not contain more than 30 characters, and each line of the specfile is limited to 80 characters, including the blanks separating keyword and value.

The portion of the specification file used by CHECK_read_specfile must start with a "BEGIN CHECK" command and end with an "END" command. The syntax of the specfile is thus defined as follows:

```
( .. lines ignored by CHECK_read_specfile .. )
  BEGIN CHECK
     keyword    value
     .......    .....
     keyword    value
  END
( .. lines ignored by CHECK_read_specfile .. )
```

All use is subject to the conditions of a BSD-3-Clause License.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

where keyword and value are two strings separated by (at least) one blank. The "BEGIN CHECK" and "END" delimiter command lines may contain additional (trailing) strings so long as such strings are separated by one or more blanks, so that lines such as

```
BEGIN CHECK SPECIFICATION
```

and

```
END CHECK SPECIFICATION
```

are acceptable. Furthermore, between the "BEGIN CHECK" and "END" delimiters, specification commands may occur in any order. Blank lines and lines whose first non-blank character is ! or * are ignored. The content of a line after a ! or * character is also ignored (as is the ! or * character itself). This provides an easy manner to "comment out" some specification commands, or to comment specific values of certain control parameters.

The value of a control parameters may be of three different types, namely integer, logical, or real. Integer and real values may be expressed in any relevant Fortran integer and floating-point formats (respectively). Permitted values for logical parameters are "ON", "TRUE", ".TRUE.", "T", "YES", "Y", or "OFF", "NO", "N", "FALSE", ".FALSE." and "F". Empty values are also allowed for logical control parameters, and are interpreted as "TRUE".

The specification file must be open for input when CHECK_read_specfile is called, and the associated device number passed to the routine in device (see below). Note that the corresponding file is REWINDed, which makes it possible to combine the specifications for more than one program/routine. For the same reason, the file is not closed by CHECK_read_specfile.

2.8.1 To read control parameters from a specification file

Control parameters may be read from a file as follows:

```
CALL CHECK_read_specfile( control, device )
```

control is a scalar INTENT(INOUT) argument of type CHECK_control_type (see Section 2.3.3). Default values should have already been set, perhaps by calling CHECK_initialize. On exit, individual components of control may have been changed according to the commands found in the specfile. Specfile commands and the components (see Section 2.3.3) of control that they affect are given in Table 2.1.

command	component of control	value type
error-printout-device	%error	integer
printout-device	%out	integer
print-level	%print_level	integer
verification-level	%verify_level	integer
f-availability	%f_availability	integer
c-availability	%c_availability	integer
g-availability	%G_availability	integer
J-availability	%J_availability	integer
H-availability	%H_availability	integer
check-gradient	%checkG	logical
check-Jacobian	%checkJ	logical
check-Hessian	%checkH	logical
deallocate-error-fatal	%deallocate_error_fatal	logical

Table 2.1: Specfile commands and associated components of control.

All use is subject to the conditions of a BSD-3-Clause License.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

device is a scalar INTENT(IN) argument of type INTEGER(ip_), that must be set to the unit number on which the specfile has been opened. If device is not open, control will not be altered and execution will continue, but an error message will be printed on unit control%error.

2.9 Information printed

If control%print_level is positive, information about the derivative verification will be printed on unit control%out, provided control%out > 0. If control%print_level $= 1$, a basic summary of the derivative checking is produced. If control%print_level $= 2$, then in addition to the above there is detailed output of the derivative verification, control parameters are printed, and basic matrix data is produced. If control%print_level $= 3$, then in addition to the above, full matrix data is printed. Finally, control%print_level ≥ 4 is used for debugging and in addition to the above also prints private data used during the verification process.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: None.

Other modules used directly: CHECK_verify and CHECK_terminate call the GALAHAD packages GALAHAD_MOP, and GALAHAD_SPACE.

Input/output: Output is under control of the arguments control%error, control%out and control%print_level.

Restrictions: nlp%n > 0 and nlp%m ≥ 0.

Portability: ISO Fortran 95 + TR 15581 or Fortran 2003. The package is thread-safe.

4 METHOD

Finite difference approximations are used to numerically "verify" requested derivatives. If verify_level $= 1$, we perform a "cheap" check of the gradient of the objective function by comparing $(f(\mathbf{x}) - f(\mathbf{x}+\alpha\mathbf{s}))/\alpha$ with $\nabla_x f(\mathbf{x})^T\mathbf{s}$ for some appropriately chosen vector $\mathbf{s}$ and scalar $0 < \alpha \ll 1$. Similarly, for the Jacobian of the constraints and the Hessian of the Lagrangian, we compare $(\mathbf{c}(\mathbf{x}) - \mathbf{c}(\mathbf{x}+\alpha\mathbf{s}))/\alpha$ to $\nabla_x\mathbf{c}(\mathbf{x})\mathbf{s}$ and $(\nabla_x L(\mathbf{x},\mathbf{y}) - \nabla_x L(\mathbf{x}+\alpha\mathbf{s},\mathbf{y}))/\alpha$ to $\nabla_{xx}L(\mathbf{x},\mathbf{y})\mathbf{s}$, respectively. If verify_level $= 2$, we perform an "expensive" verification of the gradient of the objective function by comparing $(f(\mathbf{x}) - f(\mathbf{x}+\alpha\mathbf{e}_i))/\alpha$ with $[\nabla_x f(\mathbf{x})]_i$ for $i = 1,\ldots,n$, where $\mathbf{e}_i$ is the ith coordinate vector. Similarly, for the Jacobian of the constraints and the Hessian of the Lagrangian, we compare $[(\mathbf{c}(\mathbf{x}) - \mathbf{c}(\mathbf{x}+\alpha\mathbf{e}_j))]_i/\alpha$ to $[\nabla_x\mathbf{c}(\mathbf{x})]_{ij}$ and $[(\nabla_x L(\mathbf{x},\mathbf{y}) - \nabla_x L(\mathbf{x}+\alpha\mathbf{e}_j,\mathbf{y}))]_i/\alpha$ to $[\nabla_{xx}L(\mathbf{x},\mathbf{y})]_{ij}$, respectively.

5 EXAMPLES OF USE

Suppose we wish to perform an "expensive" check of the derivatives associated with the objective function $f(\mathbf{x}) = x_1 + x_2^3/3$ and the constraint function $\mathbf{c}(\mathbf{x}) = (x_1 + x_2^2 + x_3^3 + x_3x_2^2, -x_2^4)$ at the point $\mathbf{x} = (4,3,2)$ and $\mathbf{y} = (2,3)$, with bounds $\mathbf{x}^l = (-5,-5,-5)$ and $\mathbf{x}^u = (5,5,5)$. We may use the following code:

```
! THIS VERSION: GALAHAD 4.1 - 2021-11-27 AT 13:45 GMT.
PROGRAM GALAHAD_check_example
  USE GALAHAD_SMT_double   ! double precision version
  USE GALAHAD_USERDATA_double  ! double precision version
  USE GALAHAD_NLPT_double  ! double precision version
```

All use is subject to the conditions of a BSD-3-Clause License.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

```
  USE GALAHAD_MOP_double   ! double precision version
  USE GALAHAD_CHECK_double ! double precision version
  IMPLICIT NONE
  integer, parameter :: wp = KIND( 1.0D+0 ) ! Define the working precision
  type( NLPT_problem_type ) :: nlp
  type( GALAHAD_userdata_type ) :: userdata
  type( CHECK_data_type ) :: data
  type( CHECK_control_type ) :: control
  type( CHECK_inform_type ) :: inform
  integer :: stat, Jne, Hne, m, n
  real (kind = wp), parameter :: two = 2.0_wp, three = 3.0_wp
  real (kind = wp), parameter :: four = 4.0_wp, five = 5.0_wp
  external funF, funC, funG, funJ, funH
  nlp%m   = 2 ;  nlp%n   = 3 ;  m = nlp%m    ;  n = nlp%n
  nlp%J%m = 2 ;  nlp%J%n = 3 ;  nlp%J%ne = 4 ;  Jne = nlp%J%ne
  nlp%H%m = 3 ;  nlp%H%n = 3 ;  nlp%H%ne = 3 ;  Hne = nlp%H%ne
  call SMT_put( nlp%J%id, 'Toy 2x3 matrix', stat );
  call SMT_put( nlp%J%type, 'COORDINATE', stat )
  call SMT_put( nlp%H%id, 'Toy 3x3 hessian matrix', stat );
  call SMT_put( nlp%H%type, 'COORDINATE', stat )
  allocate( nlp%G(n), nlp%C(m), nlp%X(n), nlp%X_l(n), nlp%X_u(n), nlp%Y(m) )
  allocate( nlp%J%row(Jne), nlp%J%col(Jne), nlp%J%val(Jne) )
  allocate( nlp%H%row(Hne), nlp%H%col(Hne), nlp%H%val(Hne) )
  nlp%J%row = (/ 1, 1, 1, 2 /) ;  nlp%J%col = (/ 1, 2, 3, 2 /)
  nlp%H%row = (/ 2, 3, 3 /)    ;  nlp%H%col = (/ 2, 2, 3 /)
  nlp%X = (/ four, three, two /) ;  nlp%X_l = -five ;  nlp%X_u = five
  nlp%Y = (/ two, three /)
  call CHECK_initialize( control ) ;   control%print_level = 3
  inform%status = 1
  call CHECK_verify( nlp, data, control, inform, userdata, &
                     funF, funC, funG, funJ, funH )
  call CHECK_terminate( data, control, inform )
END PROGRAM GALAHAD_check_example

SUBROUTINE funF( status, X, userdata, F )
  USE GALAHAD_USERDATA_double
  INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )
  INTEGER, INTENT( OUT ) :: status
  REAL ( kind = wp ), INTENT( IN ), DIMENSION( : ) :: X
  REAL ( kind = wp ), INTENT( OUT ) :: F
  TYPE ( GALAHAD_userdata_type ), INTENT( INOUT ) :: userdata
  F = X(1) + X(2)**3 / 3.0_wp
  status = 0
  RETURN
END SUBROUTINE funF
SUBROUTINE funC(status, X, userdata, C)
  USE GALAHAD_USERDATA_double
  INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )
  INTEGER, INTENT( OUT ) :: status
  REAL ( kind = wp ), INTENT( IN ), DIMENSION( : ) :: X
  REAL ( kind = wp ), DIMENSION( : ), INTENT( OUT ) :: C
  TYPE ( GALAHAD_userdata_type ), INTENT( INOUT ) :: userdata
  C(1) = X(1) + X(2)**2 + X(3)**3 + X(3)*X(2)**2
  C(2) = -X(2)**4
  status = 0
```

All use is subject to the conditions of a BSD-3-Clause License.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

```
  RETURN
END SUBROUTINE funC
SUBROUTINE funG(status, X, userdata, G)
  USE GALAHAD_USERDATA_double
  INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )
  INTEGER, INTENT( OUT ) :: status
  REAL ( KIND = wp ), DIMENSION( : ), INTENT( IN ) :: X
  REAL ( KIND = wp ), DIMENSION( : ), INTENT( OUT ) :: G
  TYPE ( GALAHAD_userdata_type ), INTENT( INOUT ) :: userdata
  G(1) = 1.0_wp
  G(2) = X(2)**2
  G(3) = 0.0_wp
  status = 0
  RETURN
END SUBROUTINE funG
SUBROUTINE funJ(status, X, userdata, Jval)
  USE GALAHAD_USERDATA_double
  INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )
  INTEGER, INTENT( OUT ) :: status
  REAL ( KIND = wp ), DIMENSION( : ), INTENT( IN ) :: X
  REAL ( KIND = wp ), DIMENSION( : ), INTENT( OUT ) :: Jval
  TYPE ( GALAHAD_userdata_type ), INTENT( INOUT ) :: userdata
  Jval(1) = 1.0_wp
  Jval(2) = 2.0_wp * X(2) * ( 1.0_wp + X(3) )
  Jval(3) = 3.0_wp * X(3)**2 + X(2)**2
  Jval(4) = -4.0_wp * X(2)**3
  status = 0
  RETURN
END SUBROUTINE funJ
SUBROUTINE funH(status, X, Y, userdata, Hval)
  USE GALAHAD_USERDATA_double
  INTEGER, PARAMETER :: wp = KIND( 1.0D+0 )
  INTEGER, INTENT( OUT ) :: status
  REAL ( kind = wp ), DIMENSION( : ), INTENT( IN ) :: X
  REAL ( kind = wp ), DIMENSION( : ), INTENT( IN ) :: Y
  REAL ( kind = wp ), DIMENSION( : ), INTENT( OUT ) ::Hval
  TYPE ( GALAHAD_userdata_type ), INTENT( INOUT ) :: userdata
  Hval(1) =  2.0_wp * ( X(2) - Y(1) - Y(1)*X(3) + 6.0_wp*Y(2)*X(2)**2 )
  Hval(2) = -2.0_wp * Y(1) * X(2)
  Hval(3) = -6.0_wp * Y(1) * X(3)
  status = 0
  RETURN
END SUBROUTINE funH
```

The code produces the following output:

```
 --------------------------------------------------------------------------------
 --------------------          BEGIN: CHECK_verify         --------------------
 --------------------------------------------------------------------------------

                  EXPENSIVE VERIFICATION OF THE GRADIENT G(X)

    Component      Ok      Difference            Value             Error
  -------------   ---   ----------------   ----------------   ----------------
  G(          1)   OK    9.999999891E-01    1.000000000E+00    5.437063107E-09
```

All use is subject to the conditions of a BSD-3-Clause License.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**

```
G(          2)    OK    9.000000276E+00    9.000000000E+00    2.755274155E-08
G(          3)    OK    0.000000000E+00    0.000000000E+00    0.000000000E+00

                EXPENSIVE VERIFICATION OF THE JACOBIAN C(X)

   Component      Ok      Difference           Value            Error
--------------   ---   ----------------   ----------------   ----------------
J(    1,    1)    OK    9.999999891E-01    1.000000000E+00    5.437063107E-09
J(    2,    1)    OK    0.000000000E+00    0.000000000E+00    0.000000000E+00
J(    1,    2)    OK    1.800000025E+01    1.800000000E+01    1.328101027E-08
J(    2,    2)    OK   -1.080000049E+02   -1.080000000E+02    4.540609994E-08
J(    1,    3)    OK    2.100000052E+01    2.100000000E+01    2.356511149E-08
J(    2,    3)    OK    0.000000000E+00    0.000000000E+00    0.000000000E+00

                EXPENSIVE VERIFICATION OF THE HESSIAN H(X,Y)

   Component      Ok      Difference           Value            Error
--------------   ---   ----------------   ----------------   ----------------
H(    1,    1)    OK    0.000000000E+00    0.000000000E+00    0.000000000E+00
H(    2,    1)    OK    0.000000000E+00    0.000000000E+00    0.000000000E+00
H(    3,    1)    OK    0.000000000E+00    0.000000000E+00    0.000000000E+00
H(    1,    2)    OK    0.000000000E+00    0.000000000E+00    0.000000000E+00
H(    2,    2)    OK    3.180000098E+02    3.180000000E+02    3.083032558E-08
H(    3,    2)    OK   -1.200000017E+01   -1.200000000E+01    1.294047150E-08
H(    1,    3)    OK    0.000000000E+00    0.000000000E+00    0.000000000E+00
H(    2,    3)    OK   -1.200000017E+01   -1.200000000E+01    1.294047150E-08
H(    3,    3)    OK   -2.400000049E+01   -2.400000000E+01    1.943240311E-08

            ------------------------------------------------------
            |                     SUMMARY                        |
            |             ( Verify : Expensive )                 |
            ------------------------------------------------------

            THE GRADIENT OF THE OBJECTIVE FUNCTION IS ---- [OK]

            THE JACOBIAN OF THE CONSTRAINT FUNCTION IS --- [OK]

            THE HESSIAN OF THE LAGRANGIAN FUNCTION IS ---- [OK]

          ----------------------------------------------------------
          |                CONTROL PARAMETERS                      |
          ----------------------------------------------------------

          checkG =  T   f_available =  1   deall_error_fatal =  F
          checkJ =  T   c_available =  1   print_level       =  3
          checkH =  T   g_available =  1   verify_level      =  2
          error  =  6   J_available =  1   out               =  6
                        H_available =  1

             ------------------------------------------------
             |                MATRIX DATA                   |
             ------------------------------------------------

             J%type --- COORDINATE
             J%id   --- Toy 2x3 matrix
```

All use is subject to the conditions of a BSD-3-Clause License.
See `http://galahad.rl.ac.uk/galahad-www/cou.html` **for full details.**

```
H%type --- COORDINATE
H%id   --- Toy 3x3 hessian matrix

m =      2
n =      3

  J%row         J%col             J%val
  -----         -----       -----------------
      1             1        1.0000000000E+00
      1             2        1.8000000571E+01
      1             3        2.1000001142E+01
      2             2       -1.0800000000E+02

  H%row         H%col             H%val
  -----         -----        ----------------
      2             2        3.1800000000E+02
      3             2       -1.2000000000E+01
      3             3       -2.4000000000E+01

                    EXIT STATUS :   0

------------------------------------------------------------------------------
--------------------          END: CHECK_verify          --------------------
------------------------------------------------------------------------------
```

All use is subject to the conditions of a BSD-3-Clause License.
See http://galahad.rl.ac.uk/galahad-www/cou.html **for full details.**